REI **AD-A275 431**

Form Approved
OMB No. 0704-0188

2

1. AGENCY USE ONLY	1/94
3. REPORT TYPE AND DATES COVERED	Scientific Paper
4. TITLE AND SUBTITLE	An Efficient Algorithm for 3D Connect-the-dots
5. FUNDING NUMBERS	
6. AUTHOR(S)	J.John Kim
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	U.S. Army Topographic Engineering Center ATTN: CETEC-PAO 7701 Telegraph Road Alexandria, VA 22310-3864
8. PERFORMING ORGANIZATION REPORT NUMBER	R-210
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	
10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	Approved for public release; distribution is unlimited.
12b. DISTRIBUTION CODE	

13. ABSTRACT *(Maximum 200 words)*

This paper proposes an efficient algorithm to solve the problem. In the algorithm, piecewise cubic Bezier curves will have a tangential continuity at the end points of each piece, so that the whole curve is continuous and differentiable everywhere. The curvature around a given point and the shape of the whole curve are controllable with user parameters. This paper analyzes the effects of the user parameters on the whole curve and will discuss the implementation of the algorithm in an efficient way using matrix operations.

14. SUBJECT TERMS	terrain, visualization, flight path
15. NUMBER OF PAGES	10
16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT	unclassified
18. SECURITY CLASSIFICATION OF THIS PAGE	unclassified
19. SECURITY CLASSIFICATION OF ABSTRACT	unclassified
20. LIMITATION OF ABSTRACT	

An Efficient Algorithm for 3D Connect-the-dots

J. John Kim
U.S. Army Topographic Engineering Center
Battlefield Visualization Division
Fort Belvoir, Virginia 22060-5546
(703) 355-3838
jjkim@tecsun1.tec.army.mil

\\PS

ABSTRACT

Given a sequence of points in 3D space, finding a smooth curve containing all of the points is called a "connect-the-dots" problem and is not computationally easy to solve when the curve must be continuous and differentiable. Many known algorithms produce a curve which only approximates the points, a curve which is not "smooth" enough, or a curve which takes too long to compute for use in a realtime situation such as terrain data visualization sequences.

This paper proposes an efficient algorithm to solve the problem. In the algorithm, piecewise cubic Bezier curves will have a tangential continuity at the end points of each piece, so that the whole curve is continuous and differentiable everywhere. The curvature around a given point and the shape of the whole curve are controllable with user parameters. This paper analyzes the effects of the user parameters on the whole curve and will discuss the implementation of the algorithm in an efficient way using matrix operations.

The algorithm has been designed and implemented to generate a realtime flight path which is required to pass through a given sequence of points and to give a directional vector determining yaw and pitch at any point on the path. This paper illustrates the actual 3D flight paths generated and used in a 3D terrain visualization project.

INTRODUCTION

When playing "connect-the-dots", a child can easily draw a smooth curve through dots by changing its curvature constantly, dot after dot in a sequence. This capability of connecting dots with a smooth curve in real time is often necessary in computer graphics. Given a sequence of discrete points in 3D space, generating a continuous curve that connects all the points is a well defined problem. This problem should not be confused with the problem of approximating points. The approximation problem does not require the resulting curve to pass through the points but does require the pre-determined order of the curve to be fitted *over all the data points*. On the contrary, the connect-the-dots problem requires that the curve must pass through all the points but does not have to fit all the data points at once as long as the resulting curve is smooth enough.

This paper proposes an efficient algorithm for the connect-the-dots problem. To meet the requirement, the algorithm uses a set of piecewise cubic Bezier curves having a tangential continuity at the end of each piece. Any neighboring curves shall have a common tangential value at the shared point (i.e., at the ending point of one curve and the starting point of the next curve) so that the combined curve is differentiable everywhere.

A cubic Bezier curve can be expressed as a function of a set of four points and derived from the general cubic curve equation over t as shown below.

$$Cubic(t) = at^3 + bt^2 + ct + d, \qquad 0 \le t \le 1$$

The above equation can be written as a vector product:

$$Cubic(t) = [t^3\ t^2\ t\ 1]\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

The coefficient column matrix determines the shape of the curve and can be defined as a function of a set of four control points {p1 p2 p3 p4} as expressed below:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = M \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \end{bmatrix}$$

where the matrix M will characterize the relationship between the curve shape and the control points. The matrix M is called the *basis* of the curve. The cubic curves can be classified on the basis of the *bases*. Cubic Bezier curves are one such class. A cubic Bezier curve passes through the first and fourth control points and has the slopes determined by the second and third control points. This property of a Bezier curve can be written mathematically as follows:

$$\begin{aligned} Bezier(0) &= p1 \\ Bezier(1) &= p4 \\ Bezier'(0) &= 3(p2 - p1) \\ Bezier'(1) &= 3(p4 - p3) \end{aligned}$$

From the above set of equations, the Bezier basis M is derived.

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The Bezier cubic curve is now expressed as a function of only a set of four control points over t.

$$\mathit{Bezier}(t) = [t^3\ t^2\ t1\ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \end{bmatrix}, \qquad 0 \le t \le 1$$

This Bezier curve is applied to the control points given as 3D dots in the "connect-the-dots" problem. The application details and the control mechanism over the shape of the curve are described in the next two sections. The last two sections show implementation details and the illustrations of the curve as a smooth flight path in a 3D terrain visualization system.

PIECEWISE BEZIER CURVE

In generating a curve that passes through a long sequence of control points, it is practical to make the curve a piecewise polynomial because such a curve is easy to manipulate. Among many different piecewise polynomial curves, a piecewise Bezier curve was chosen to meet the requirement of the "connect-the-dots" problem. A piecewise Bezier cubic curve requires a given sequence of control points to be grouped into a sequence of sets of four control points to which the Bezier equation derived in the previous section applies. The grouping should be done carefully, though. Otherwise, neighboring pieces of Bezier curves would not meet at the same slope and result in a not-so-smooth piecewise Bezier curve. To remedy the problem, it is possible to use overlapped sets of four control points such as {a1 a2 a3 a4}, {a3 a4 a5 a6}, {a5 a6 a7 a8}, and so on. The overlapping grouping will neither generate a curve passing through all points nor give room for controlling the behavior of the resulting curve. They may also be grouped unevenly.

This paper proposes that two extra points be inserted between every two consecutive control points in the sequence such that any control point and two extra points newly inserted before and after the control point become collinear (See Figure 1). In the figure, the quartets (each of which defines a Bezier curve) are Control Point i, Control Point i+1, and two inserted points between, i = 1, 2, 3, 4, and 5. The extra points not only make a smooth piecewise Bezier curve passing through all the given control points but also provide the leverage to control the shape of the curve, which will be discussed in the next section.

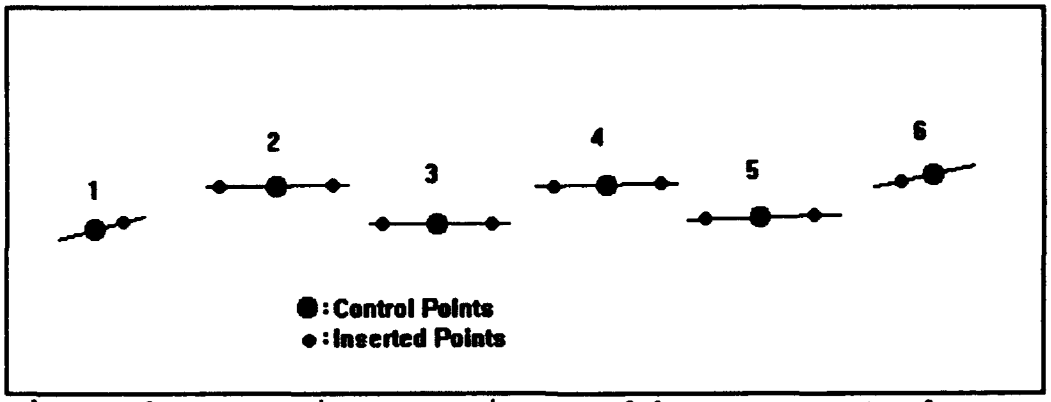

Figure 1. Two points are inserted between control points. A control point and its nearby inserted points are collinear.

CONTROL OF CURVE SHAPE

The manipulation of the inserted points controls the curvature and the slope of a piece of Bezier curve independently. The position and orientation of the inserted points mostly affect the behavior of the piecewise Bezier curve constructed by the method described above. Consider a sequence of given control points denoted by C and inserted points by i as described in the earlier section:

$$C_1,\ i_2,\ i_3,\ C_4,\ i_5,\ i_6,\ C_7,\ i_8,\ i_9,\ C_{10},\ i_{11},\ \ldots$$

The distance between the inserted points and a control point, say i_3 and C_4, determines the curvature of the incoming curve into the control point C_4. It is intuitive to see that the closer the inserted points i_3 and i_5 are located to the control point C_4 the smaller the curvature of the curve around C_4 is. This intuition is well illustrated in Figure 2.

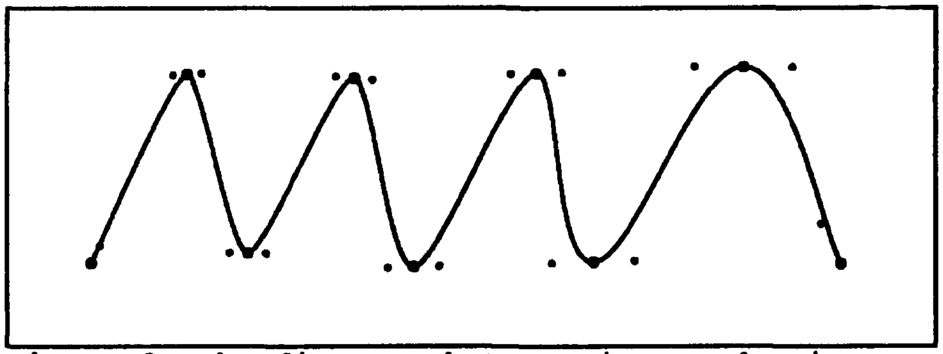

Figure 2. The distance between inserted points (smaller dots) and a control point affects the local curvature around that control point.

It may be practical to maintain equal distances between the inserted points and a control point so that the incoming and outgoing curves at the control point have the same curvature. The orientation of the line consisting of a control point and two inserted points is another controlling factor. Let's revisit the above sequence. The slope at C_4 of the Bezier curve for the set of four points $\{C_1\ i_2\ i_3\ C_4\}$ is determined by the difference of i_3 and C_4, and the slope at C_4 of the curve for the next set $\{C_4\ i_5\ i_6\ C_7\}$ by the difference of i_5 and C_4. The two slopes at C_4 must be the same since the inserted points are placed to make the points $i_3C_4i_5$ collinear. Both the incoming and outgoing curves at C_4 have a common tangential line, which is the line $i_3C_4i_5$. The setup of collinearity and equal distance of inserted points around a control point guarantees a continuous and differentiable curve and leaves the orientation of the collinear line and the distance of the inserted points as input parameters to control the behavior of the curve.

IMPLEMENTATION

The implementation of the Bezier curve is straightforward. The traditional forward difference algorithm is very effective for computing any number of discrete points in a Bezier curve successively. This section describes in detail the implementation of a forward difference matrix for a Bezier curve.

A curve is regarded as an infinite number of sequenced points. What is actually needed from the piecewise Bezier curve is enough sequenced points to reconstruct the curve as closely as an application requires. The number of required sequenced points varies depending on an application. For example, drawing a piecewise Bezier curve as a solid curvy line on a graphics monitor requires fewer points than using the same curve as a 3D flight path in a mission rehearsal. Any number of points can be generated by evaluating Bezier(t) for 0<=t<=1. If n denotes the number of points, Bezier(t) will be evaluated n+1 times for the following t values:

$$t = 0,\ \frac{1}{n},\ \frac{2}{n},\ \ldots,\ \frac{n}{n}$$

The points generated will be as follows:

$$Bezier(0) = [0\ 0\ 0\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}p1\\ p2\\ p3\\ p4\end{bmatrix}$$

$$Bezier(\frac{1}{n}) = \left[(\frac{1}{n})^3\ (\frac{1}{n})^2\ \frac{1}{n}\ 1\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}p1\\ p2\\ p3\\ p4\end{bmatrix}$$

$$Bezier(\frac{2}{n}) = \left[(\frac{2}{n})^3\ (\frac{2}{n})^2\ \frac{2}{n}\ 1\right]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}p1\\ p2\\ p3\\ p4\end{bmatrix}$$

$$\vdots$$

$$Bezier(\frac{n}{n}) = [1\ 1\ 1\ 1]\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 3 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}p1\\ p2\\ p3\\ p4\end{bmatrix}$$

Each of the n points can be computed easily, when the forward difference algorithm is used, by one iteration of adding the third row to the fourth row, the second to the third, and the first to the second in a certain matrix. Such a matrix is called a forward difference matrix and it is not difficult to construct. Let's consider the matrix F shown below:

$$F = \begin{bmatrix}\frac{6}{n^3} & 0 & 0 & 0\\ \frac{6}{n^3} & \frac{2}{n^2} & 0 & 0\\ \frac{1}{n^3} & \frac{1}{n^2} & \frac{1}{n} & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

The bottom row of the above matrix F initially gives a quartet to be used as the first factor in the Bezier equation for the computation of the first point of the n+1 points to be generated. After one iteration of adding the third row to the fourth row, the second to the third, and the first to the second of F, the bottom row of the updated matrix gives another quartet to be used as the first factor in the Bezier equation for the computation of the second point. And this iteration goes on until the n-th iteration produces the last quartet in the bottom row for the last point of the n+1 points. The following sequence shows the n+1 quartets generated by the successive iteration of the matrix F.

$$(0,0,0,1);\ ((\frac{1}{n})^3,(\frac{1}{n})^2,\frac{1}{n},1);\ ((\frac{2}{n})^3,(\frac{2}{n})^2,\frac{2}{n},1);\ \ldots;\ (1,1,1,1$$

To make it even simpler, a forward difference matrix is constructed as the product of matrices F, M, and a single column matrix of [p1 p2 p3 p4] as shown below. The resulting 4X4 matrix is called a Bezier forward difference matrix.

$$F_{Bezier} = \begin{bmatrix} \frac{6}{n^3} & 0 & 0 & 0 \\ \frac{6}{n^3} & \frac{2}{n^2} & 0 & 0 \\ \frac{1}{n^3} & \frac{1}{n^2} & \frac{1}{n} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p1 \\ p2 \\ p3 \\ p4 \end{bmatrix}$$

The above Bezier forward difference matrix will generate a sequence of n+1 points between the points p1 and p4 with the first in the sequence being P1 and the last p4. In our application, p1 and p4 can be any neighboring control points in the given input sequence, and p2 and p3 are the inserted extra points between the neighboring control points. It should be noted that the extra points inserted may or may not be among the sequence of n+1 points.

FLIGHT PATH APPLICATION

Using piecewise Bezier curves for solving "connect-the-dot" problems is useful for many application areas. In this section, the curves are illustrated as flight paths used in a 3D terrain visualization system.

A smoothly generated flight path provides a smooth sequence of view points in a 3D visualization, where abrupt positional changes of the view point are undesirable. The directional derivatives at any point in the path show the change rates of yaw, pitch and orientation at that point quantitatively.

In the flight path examples (Figure 3), first, the inserted points (denoted by i's) are placed such that the collinear points $i_{k-1}C_ki_{k+1}$ are parallel to the line $C_{k-3}C_{k+3}$ except when C_k is the first or the last control point. At the first control point C_1, the inserted point i_2 is placed to make the line C_1i_2 parallel to the line C_1C_3, and at the last control point C_{last}, the inserted point i_{last-1} is placed to make the line $i_{last-1}C_{last}$ parallel to the line $C_{last-3}C_{last}$. The orientation of the collinear points shown in Figures 1 and 2 of the earlier sections illustrates this setup. Second, the distances between inserted points and control points are locally adjusted interactively before a control point is selected and marked on a terrain image (or a map). Since the distance controls the curvature of a flight path, the minimum distance could depend on the capability of a specific aircraft. Third, the number of samples taken from each piece of Bezier curve is dynamically adjusted so that the distance between any two samples, called frame distance, is maintained constant through out the whole curve. Typical frame distances for a flight path application are 10, 20, 30, 40, and 50 meters. For example, a 20 meter frame distance gives a new aircraft position (or a new view point) every 20 meters along the whole flight path. This means a new 3D perspective terrain scene is automatically redrawn on the screen every 20 meters when the auto-flight mode is chosen in a 3D terrain visualization system. The following figure shows three flight paths drawn on three different terrain (SPOT image or map) backgrounds, where the altitudes of the paths are drawn along the cross-sectional view of the terrain at the bottom of the map. The altitude of a control point is set to a constant value above the corresponding terrain elevation to simplify the illustration.

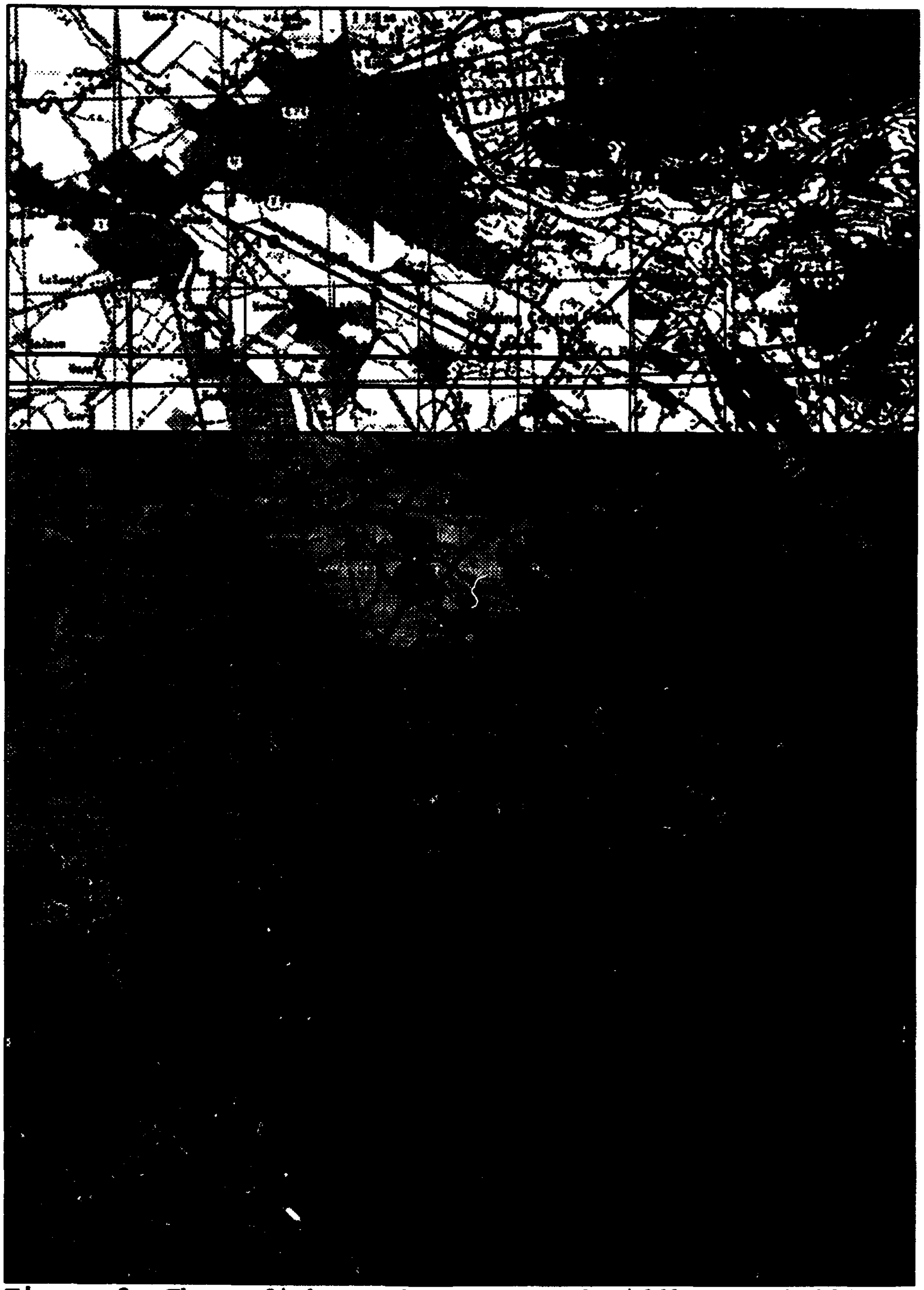
Figure 3. Three fight paths - Top and Middle: 1:50,000 and 1:250,000 scale maps of Sarajevo area, Bottom: SPOT image of Foca area

CONCLUSION

This paper has presented an efficient method to solve the connect-the-dots problem computationally. The method systematically inserts extra points in a given sequence of dots (control points) and applies a Bezier cubic curve to each quartet of points in the sequence to get a piecewise Bezier curve. The piecewise Bezier curve is guaranteed to have a directional derivative anywhere by the systematic insertion of the extra points. The inserted points were shown to control the behavior of the curve. The method was applied to flight path generation and is empirically proven to be robust.